UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
ESCUELA DE CIENCIAS Y SISTEMAS
LENGUAJES FORMALES Y DE PROGRAMACIÓN

Primer Semestre 2020

Sección	Catedrático	Tutor académico
A-	Inga. Damaris Campos de López	Fernando Mérida
B-	Inga. Zulma Aguirre	Luis Javier Yela Quijada
A+	Ing. Otto	Elmer Real
B+	Ing.	José Veliz

Enunciado de Práctica

Objetivos:

- Que el estudiante implemente una solución de software con base en los distintos paradigmas de programación vistos en clase y laboratorio.
- Conocer y aprender el lenguaje de programación Python.

Descripción:

Una empresa dedicada al desarrollo de software le solicita a usted, estudiante del curso de Lenguajes Formales, desarrollar un programa en consola multiproposito.
Esta aplicación denominada EasyGames tiene diferentes propositos. Estos son:

- Entretenimiento
- Educación

En la categoría de entretenimiento

- Se desarrollara un programa utilizando el paradigma orientado a objetos que permita crear mascotas virtuales e interactuar con ellas.

En la categoría de educación

- Se desarrollaran diferentes funcionalidades que permitan practicar matemáticas
- Se desarrollara un almacén de caracteres que permita manejar cadenas largas.

Para poder hacer uso de las funcionalidades desarrolladas se debe de implementar la lectura de archivos de texto con extension **.mascota**, **.edu** y **.almacen**. Las cuales contendran una conjunto de instrucciones especificas, mas adelante se explicara de manera explicita cuales son y el comportamiento esperado.

A grandes rasgos cada linea en un archivo de texto extension .**mascota, .edu o .almacen** tendrán el siguiente patrón.

<NombreInstruccion>:<Parametro1>,<Parametro2>...<ParametroN>

- La cantidad de parámetros depende de la instrucción.
- No se recomienda que implemente un analizador para reconocer cada componente de las instrucciones..

Ejemplo:

Crear_Pajaro:Pajaro1

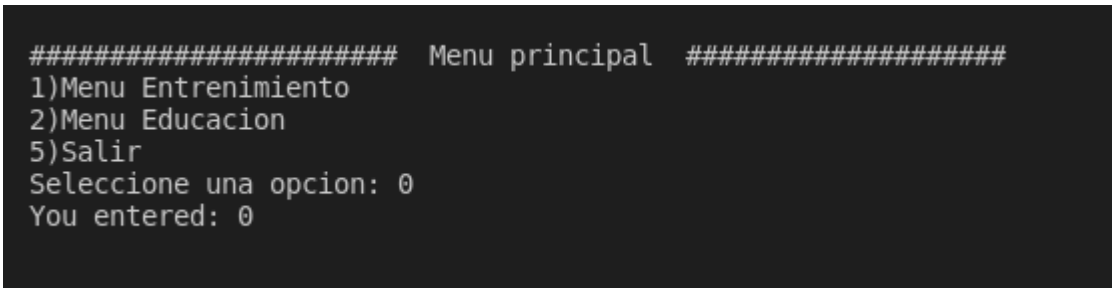
Características del programa

- **Menú Principal**
  - Deberá de ser una caratula en donde se indique el nombre del curso, la sección, y su carné, al presionar la tecla enter deberá de dirigirse a otra pantalla en la cual habrá un menú con las opciones:

    - Menú Entretenimiento: Limpiara la pantalla y nos mostrara la opciones disponibles para entretenimiento.
    - Menú Educación: Limpiara la pantalla y nos mostrara la opciones disponibles para educación
    - Salir: Terminara la aplicación.

- **Menú Entretenimiento**
  - Cargar Archivo con extensión ***.mascotas***
    - Limpia el arreglo de mascotas existentes y ejecuta cada una de las instrucciones contenidos en el archivo. Deberá de generar un archivo ***.mascotas_result***, mas adelante se le indicara que debe de imprimir en este archivo.
  - Regresar a pantalla inicial.
    - Regresar a pantalla inicial.

- **Menú Educación**
  - Cargar Archivo con extensión ***.edu***
    - Limpia el arreglo de mascotas existentes y ejecuta cada una de las instrucciones contenidos en el archivo. Deberá de generar un archivo ***.edu_result***, mas adelante se le indicara que debe de imprimir en este archivo.
  - Cargar Archivo con extensión ***.almacen***
    - Limpia el arreglo de mascotas existentes y ejecuta cada una de las instrucciones contenidos en el archivo. Deberá de generar un archivo ***.almacen_result***, mas adelante se le indicara que debe de imprimir en este archivo.
  - Regresar a pantalla inicial.



Mascotas Virtuales utilizando POO:

Existen dos tipos de mascotas: Pájaro y Gato. Tenga en cuenta que si después de comer un ratón o ir a dejar una mensaje la mascota se queda sin energía se muere.

El pájaro:

- Cuando vuela, consume un joule de energía para cada kilómetros que vuela, mas 10 joules que necesita para alzar vuelo.
- Cuando come , adquiere 4 joules de energía por cada gramo que come.
- Para volar se le debe de indicar el punto cartesiano (x,y) al que se dirige. El pájaro vuela directamente a su destino por lo que la distancia entre los dos puntos es la distancia de kilómetros que vuela en total.
- Cuando se crea la mascota su posición inicial sera (0,0) y estará vivo.

El gato:

- Cuando corre su energía disminuye la mitad de metros que recorrió.
- Cuando come, su energía aumenta en 12 joules + el peso del ratón en gramos.
- Para comer un ratón, se le debe de indicar el punto cartesiano (x,y) en el que esta el ratón. El gato corre directamente a su destino por lo que la distancia entre los dos puntos es la cantidad de metros que corre en total.

Redondear al entero mas cercano en caso la distancia calculada tenga decimales.

Archivo .mascotas

Este archivo tendrá diferentes comandos utilizados para crear, interactuar y manipular a las mascotas.

- **Crear_Pajaro:<Nombre>**
  - Esta instrucción crea una mascota de tipo pájaro.
    - En el archivo .mascotas_result imprimirá:
      - [Fecha hora] Se creo el pájaro <Nombre>
- **Puede_Entregar_Mensaje:<Nombre>,<CoordenadaX>,<CoordenadaY>**
  - Esta instrucción aplica solo a pájaros y sirve para verificar si tiene la energía suficiente para ir a dejar el mensaje.
    - En el archivo .mascotas_result imprimirá:
      - Si le alcanza la energía:
        - [Fecha hora] <Nombre>, Si puedo ir a dejar el mensaje
      - De lo contrario:
        - [Fecha hora] <Nombre>, Estoy exhausto. Dame de comer <# Gramos> para ir.
      - Si se murió:
        - [Fecha hora] <Nombre>, ya me mori.
- **Enviar_Mensaje:<Nombre>,<CoordenadaX>,<CoordenadaY>**
  - Esta instrucción aplica solo a pájaros y esta cambiara las coordenadas del pájaro a las indicadas y modificara su energía.
    - En el archivo .mascotas_result imprimirá:
      - Si le alcanza la energía:
        - [Fecha hora] <Nombre>, Ya fui a dejar el mensaje a (<X>,<Y>)
      - De lo contrario:
        - [Fecha hora] <Nombre>, Estoy exhausto. Dame de comer <# Gramos> para ir.
      - Si ya se murió:
        - [Fecha hora] <Nombre>, ya me mori.

- **Crear_Gato:<Nombre>**
  - Esta instrucción crea una mascota de tipo gato.
    - En el archivo .mascotas_result imprimirá:
      - [Fecha hora] Se creo el gato <Nombre>
- **Conviene_Comer_Raton:<Nombre>,<CoordenadaX>,<CoordenadaY>,<Peso>**
  - Esta instrucción indica aplica solo para gatos. Verifica si la energía gastada se compensa por la energía que recibirá al comer al raton. No modifica el estado del gato.
    - En el archivo .mascotas_result imprimirá:
      - Si le conviene
        - [Fecha hora] <Nombre>, Si me conviene comerme al gato
      - De lo contrario
        - [Fecha hora] <Nombre>, Esta muy lejos. No me conviene.
      - Si ya se murió:
        - [Fecha hora] <Nombre>, ya me mori.

- **Enviar_Comer_Raton:<Nombre>,<CoordenadaX>,<CoordenadaY>,<Peso>**
  - Esta instrucción indica aplica solo para gatos. Enviá al gato a la dirección indicada para comerse un ratón.
    - En el archivo .mascotas_result imprimirá:
      - Si le alcanza la energía:
        - [Fecha hora] <Nombre>, Ya me comí al ratón, ahora mi energía es <Energia>
      - De lo contrario:
        - [Fecha hora] <Nombre>, Estoy exhausto. Dame de comer <# Gramos> para ir.
      - Si ya se murió:
        - [Fecha hora] <Nombre>, ya me morí.
- **Dar_de_Comer: <Nombre>,<Peso>**
  - Esta instrucción indica para cualquier mascota. Aumenta la energía dependiendo de que mascota es.
    - En el archivo .mascotas_result imprimirá:
      - Si esta vivo
        - [Fecha hora] <Nombre>, Gracias. Ahora mi energia es <Energia>
      - Si esta muerto:
        - [Fecha hora] <Nombre>, Muy tarde. Ya me morí.
- **Resumen_Mascota:<Nombre>**
  - Esta instrucción indica para cualquier mascota.
    - En el archivo .mascotas_result imprimirá:
      - [Fecha hora] <Nombre>, Energia: <Energia>, X: <X>, Y: <Y>, <Tipo>, <Estado>
- **Resumen_Global**
  - Esta instrucción imprimirá el estado de cada mascota en el archivo .mascotas_result
    - Ejemplo:
      - [Fecha hora] ---------- Resumen Global ----------
      - [Fecha hora] <Nombre>, Energia: <Energia>, X: <X>, Y: <Y>,<Tipo>,<Estado>
      - [Fecha hora] <Nombre>, Energia: <Energia>, X: <X>, Y: <Y>, <Tipo>,<Estado>

- ….
- [Fecha hora] ----------

- Los parámetros que se utilizaran en las sentencias serán de tipo:
  - Cadena
    - Nombre
  - Enteros
    - CoordenadaX
    - CoordenadaY

Ejemplo:

archivo.mascotas

```
Crear_Pajaro:"Peter"
Crear_Gato:"Miau"
Dar_de_Comer:"Peter",800
Dar_de_Comer:"Miau",700
Puede_Entregar_Mensaje:"Peter",10,10
Puede_Entregar_Mensaje:"Miau",10,10
Enviar_Mensaje:"Peter",10,10
Resumen_Mascota:"Peter"
Conviene_Comer_Raton:"Miau",20,20,5
Conviene_Comer_Raton:"Miau",10,10,25
Enviar_Comer_Raton:"Miau",10,10,25
Resumen_Mascota:"Miau"
Resumen_Global
```

archivo.mascotas_result

```
[12/02/2020 13:00] Se creo el pájaro Peter
[12/02/2020 13:01] Se creo el gato Miau
[12/02/2020 13:01] Peter, Gracias. Ahora mi energia es 800
[12/02/2020 13:01] Miau, Gracias. Ahora mi energia es 700
[12/02/2020 13:01] Peter, Si puedo ir a dejar el mensaje
[12/02/2020 13:01] Peter,  Ya fui a dejar el mensaje a (10,10)
[12/02/2020 13:01] Peter, Energia: 786, X:10, Y:10, Pajaro, vivo
[12/02/2020 13:01] Miau, Esta muy lejos. No me conviene.
[12/02/2020 13:01] Miau, Si me conviene comerme al raton
[12/02/2020 13:02] Miau, Energia: 700, X:0, Y:0, Gato, vivo
[12/02/2020 13:02] -------------------- Resumen Global ------------
[12/02/2020 13:01] Peter, Energia: 786, X:10, Y:10, Pajaro, vivo
[12/02/2020 13:02] Miau, Energia: 700, X:0, Y:0, Pajaro, vivo
[12/02/2020 13:02] ----------------------------------------------
```

Archivo .edu

Se simulara una calculadora científica que permite almacenar variables y realizar operaciones aritméticas. El reto consiste en practicar el paradigma funcional programando un método que reciba la cadena a operar y utilizando el método de pila retorne el resultado. Cada una de los siguientes comandos deberán hacer uso, internamente, del método que usted programo, **esa es la única forma de operar las expresiones aritméticas.**

- Variable: <Nombre>,<ExpresionMatematica>
  - Guarda en un arreglo la variable junto al valor devuelto al mandar a evaluar la expresión aritmética.
- Postfija: <ExpresionMatematica>
    - En el archivo .edu_result imprimirá el postfijo de la expresión dada (cada elemento debe estar separado por un espacio). Ejemplo:
      - Postfija:5 + 3 * 8 + 20
        - [Fecha hora] Postfija: 5 3 + 8 20 + *
- Incrementar:<Nombre>
  - Aumenta el valor de la variable en 1.
- Mitad:<Nombre>
  - Disminuye el valor de la variable a la mitad.
- Triple:<Nombre>
  - Modifica la variable por 3.
- Positivo:<Nombre>
  - Convierte y modifica el valor de la variable en positivo
- Negativo:<Nombre>
  - Convierte y modifica el valor de la variable en positivo
- Potencia: <ExpresionMatematica> , <ExpresionMatematica>
  - Eleva la expresion 1 a la expresion 2.
    - En el archivo .edu_result imprimirá
      - [Fecha hora] Potencia: <Resultado>
- Imprimir: <Nombre>
  - [12/02/2020 13:00] Este es el valor de AImprime el valor de la variable.
    - En el archivo .edu_result imprimirá
      - [Fecha hora] <Nombre>, <Valor>
- Imprimir_Mensaje: <Cadena>
  - Imprime una cadena.
    - En el archivo .edu_result imprimirá
      - [Fecha hora] <Cadena>
- Las Expresiones Matemáticas se deben de operar utilizando el método de pila.
- Las expresiones aritméticas serán introducidas de manera infija por lo que internamente debe operar utilizando las reglas del método de pila.
- Cada operando y operador de las expresiones aritméticas siempre estarán separadas por un espacio.
- Ejemplo: 5 + 23 * 2
- Las operadores permitidos serán solamente (ordenados de menor a mayor jerarquía):
  - +, -
  - *,/
  - (,)

Ejemplo:

archivo.edu

```
1   Variable:A,50 + 25 * 3
2   Variable:B,1
3   Postfija:50 + 25 * 3
4   Incrementar:A
5   Mitad:A
6   Triple:B
7   Positivo:B
8   Negativo:B
9   Potencia:2 + 3, 1 * 2
10  Imprimir_Mensaje:"Este es el valor de A"
11  Imprimir:A
12  Imprimir_Mensaje:"Este es el valor de B"
13  Imprimir:B
```

archivo.edu_result

```
1   [12/02/2020 13:00] Postfijo: 50 25 3 * +
2   [12/02/2020 13:00] Potencia: 25
3   [12/02/2020 13:00] Este es el valor de A
4   [12/02/2020 13:00] 63
5   [12/02/2020 13:00] Este es el valor de B
6   [12/02/2020 13:00] -3
```

Almacén de Caracteres.

Un almacén de caracteres es un vector dinámico en el cual se almacenara una cadena carácter por carácter.
Se utilizara un identificador para saber en que posición esta almacenada cada palabra.

- Nos referiremos como almacén, al vector donde se almacenaran los caracteres de las cadenas
- Se tiene una matriz, nos referiremos a ella como tabla de variables, donde podremos almacenar el nombre y posición de cada cadena dentro del almacén.
- Al cargar un archivo .almacen se debe de limpiar el almacén y la tabla de variables.
- La posición 0 del almacén sera un numero al cual nos referiremos como puntero. El puntero indica la proxima posición libre en el almacén. Al inicio puntero=1.
- Cuando se declara una variable deberá:
  - Insertar a la tabla de variables el id y el valor del puntero.
  - Guardar en el almacen
    - La cantidad de caracteres
    - Cada carácter de forma individual.
  - Actualizar el puntero para no sobreescribir cadenas.

Archivo .almacen

- Declarar:<Nombre>,<Cadena>
- Concatenar:<Nombre>,<Nombre>
  - Copia las dos las dos palabras al final del almacén.
- Posición_cadena:<Nombre>
  - Imprime en el archivo .almacen_result la posición de la cadena :
    - [Fecha hora] <Nombre>, Pos: <Posicion>
- Tamanio:<Nombre>
  - Imprime en el archivo .almacen_result el tamaño de la cadena :
    - [Fecha hora] <Nombre>, Tam: <Tamaño>
- Imprimir:<Nombre>
  - Imprime en el archivo .almacen_result cada uno de los caracteres de la cadena :
    - [Fecha hora] <Nombre>,<Posicion>,<Caracter1>
    - [Fecha hora] <Nombre>,<Posicion> ,<Caracter2>
    - ….
    - [Fecha hora] <Nombre>,<Posicion> ,<CaracterN>

- Generar_grafo:<Cadena>
  - Genera una imagen .jpg del estado del almacén en la ruta indicada. Esta imagen se debe de realizar utilizando la libreria graphviz.

Ejemplo:

archivo.almacen

```
1  Declarar:Palabra1,"Hola"
2  Declarar:Palabra1,"Adios"
3  Posicion_cadena:Palabra1
4  Tamanio:Palabra1
5  Imprimir:Palabra1
6  Posicion_cadena:Palabra2
7  Tamanio:Palabra2
8  Imprimir:Palabra2
9  Generar_grafo:"C://Grafo"
```

archivo.almacen_result

```
1  [12/02/2020 13:00] Palabra1,1
2  [12/02/2020 13:00] Palabra1,Pos: 1
3  [12/02/2020 13:00] Palabra1,Tam: 4
4  [12/02/2020 13:00] Palabra1,2,H
5  [12/02/2020 13:00] Palabra1,3,O
6  [12/02/2020 13:00] Palabra1,4,L
7  [12/02/2020 13:00] Palabra1,5,A
8  [12/02/2020 13:00] Palabra2,Pos: 1
9  [12/02/2020 13:00] Palabra2,Tam: 4
10 [12/02/2020 13:00] Palabra2,7,A
11 [12/02/2020 13:00] Palabra2,8,D
12 [12/02/2020 13:00] Palabra2,9,I
13 [12/02/2020 13:00] Palabra2,10,O
14 [12/02/2020 13:00] Palabra2,11,S
```

ID	Posición
Palabra1	1
Palabra2	6

0) 12
1) 4
2) H
3) O
4) L
5) A
6) 5
7) A
8) D
9) I
10) O
11) S

Entregables:

- Manual de Usuario
- Manual Técnico, debe explicar con un modelo la lógica de su programa.
- Código Fuente
- Ejecutable de la Aplicación

Documentación a entregar de forma física el día de la calificación:

- Hoja de calificación (Original y una copia)

Notas importantes:

- La práctica se debe desarrollar de forma individual.
- Esta práctica se deberá desarrollar utilizando Python.
- La entrega se realizará en Classroom
- Se debe de hacer uso de los paradigmas de programación vistos en clase y laboratorio.
- No se puede agregar o quitar algún símbolo en el archivo de entrada. El proyecto deberá funcionar con los archivos de prueba que se disponga para la calificación, sin modificación.
- La calificación de la práctica será personal y durará como máximo 15 minutos, en un horario que posteriormente será establecido. Se debe tomar en cuenta que durante la calificación no podrán estar terceras personas alrededor, de lo contrario no se calificará la práctica.
- No se dará prórroga para la entrega de la práctica.
- **Copia parcial o total de la práctica tendrá una nota de 0 puntos, y se notificará a la escuela de sistemas para que se apliquen las sanciones correspondientes.**
- **En el caso de no cumplir con alguna de las indicaciones antes mencionadas, NO se calificará la practica; por lo cual, se tendrá una nota de cero puntos.**

Fecha de entrega: 02 de marzo de 2020 antes de las 23:59.